Evaluating the Use of NFTs using Event Tickets

Johnny O'Keeffe

Faculty of Science and Engineering

Department of Computer Science and Information Systems

University of Limerick

Submitted to the University of Limerick for the degree of

Final Year Project 2022

1. Supervisor: Dr. Andrew LeGear
Horizon Fintex
University *of* Limerick
Ireland

2. Supervisor: Prof. Jim Buckley
Department *of* Computer Science and Information Systems
Limerick
Ireland

Abstract

In the past year, Non-Fungible Tokens (NFTs) based on the blockchain have taken the world by storm. However, NFTs are often misunderstood. People believe they are artworks that sell for incomprehensible amounts of money. Others believe it is a scam, and retrospectively that might be the case. However, What is not highlighted is the groundbreaking blockchain technology that NFTs are using that has the potential to revolutionise digital ownership.

The purpose of this paper is to provide the reader with an insight into NFT technology from a non-blockchain perspective by investigating how NFTs can be utilized to address prominent issues in an everyday domain such as fraud in the ticketing industry. To accomplish this, we created our own event ticketing solution using NFTs as event tickets and an accompanying user interface detailing each step in the research, design and implementation of this novel solution. This paper aims to provide the reader with a better understanding of the differences between current technology and blockchain technology, what is involved in development using the blockchain, and how the blockchain can be used to upgrade and resolve issues in the ticketing industry.

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other Irish or foreign examination board.

The thesis work was conducted from year to year under the supervision of Dr. Andrew Le Gear and Prof. Jim Buckley at University of Limerick.

Limerick, 2022

Acknowledgements

Thank you to my supervisors, Dr, Andrew Le Gear and Prof. Jim Buckley, for providing continuous guidance and feedback throughout this project.

This FYP is dedicated to Lexie.

Contents

List of Figures

1

Introduction

Tickets and assigned seating for events were practices that, according to archaeological evidence, were invented by the Romans during the first century (James T. Reese and Thomas 2013). The ticketing industry has undergone significant transformations since these times, particularly with the advent of the internet as a ticket purchasing medium and as mobile tickets have seen significant uptake. But this has not been without its problems. Ticketmaster recently introduced a new technology, SafeTix, as part of its efforts to combat fraudsters and scalpers (PYMNTS 2019). Despite these efforts, issues like this still exist within the industry.

In 2018, approximately 12 percent of people reported purchasing concert tickets online which turned out to be scams (Leonhardt 2018). A more recent report from June 2021 by Action Fraud which is the UK's national reporting centre for fraud and cyber-crime received 374 reports of ticket fraud with victims reportedly losing over £200,000 pounds in March alone (Actionfraud 2021). Considering these figures, we can start to understand the magnitude of the ticket industry's problems. As technology remains the same, cybercriminals become more and more creative in their efforts to extort billions of dollars from victims desperate to attend events. During COVID-19, people are more inclined to try to obtain concert tickets since the number of events and corresponding attendees is limited, creating a situation in which fraudsters can take even more advantage. Non-Fungible Tokens (NFTs), created on a blockchain, provide a solution towards ending these problems by being verifiably original, tracking provenance, creating

a secure marketplace, and enhancing the experience for the ticketing industry with added benefits such as being able to communicate with all ticket holders and change event details on the fly. This project is novel in that it will derive an openly-available NFT solution for this problem domain, thus illustrating how this can be done.

1.1 Web3 Components

Web2 refers to the version of the internet most of us know today; An internet dominated by companies that provide services in exchange for personal data. Web3, in the context of Ethereum, refers to decentralised apps that run on the blockchain. These are apps that allow anyone to participate without monetising their personal data (Ethereum 2022*a*). In other words, Web3 refers to apps built using blockchain technology, and Web2 refers to the current technology that we use today. This section aims to present the reader different components that constitute Web3 and are essential to understanding the paper going forward. But more Web3 components will be revealed throughout the paper.

1.1.1 Blockchain

A blockchain refers to a public distributed ledger that is run by a peer to peer network of computers (see figure 1.1). Each of the computers connected holds an identical copy of the ledger, which forms a consensus of the state of the blockchain to verify the validity of cryptocurrency transactions, with rewards for providing computing power to validate the transactions. Transactions are write methods executed on the blockchain, like sending cryptocurrency to another address.

Blockchains are immutable by default meaning that once something is recorded on a blockchain it cannot be deleted, this is useful as you have access to all the activity on a blockchain to help in tracing transactions to see the source of where a particular asset or NFT came from, and assets on the blockchain cannot be deleted so they can be stored them safely. Blockchains are secure as it is impossible to tamper with them due to this design. Verifiability and immutability make a blockchain system ideal for storing digital assets.

Bitcoin's blockchain is built for the sole purpose of sending and receiving bitcoins, a digital currency known as cryptocurrency. Ethereum's blockchain implementation is similar except that it uses their own cryptocurrency called Ether and it has the addition of smart contracts, which is code hosted on the blockchain. Ethereum allows for creating other cryptocurrencies, otherwise known as tokens, and NFTs, through their smart contracts (BlockchainHub 2022).

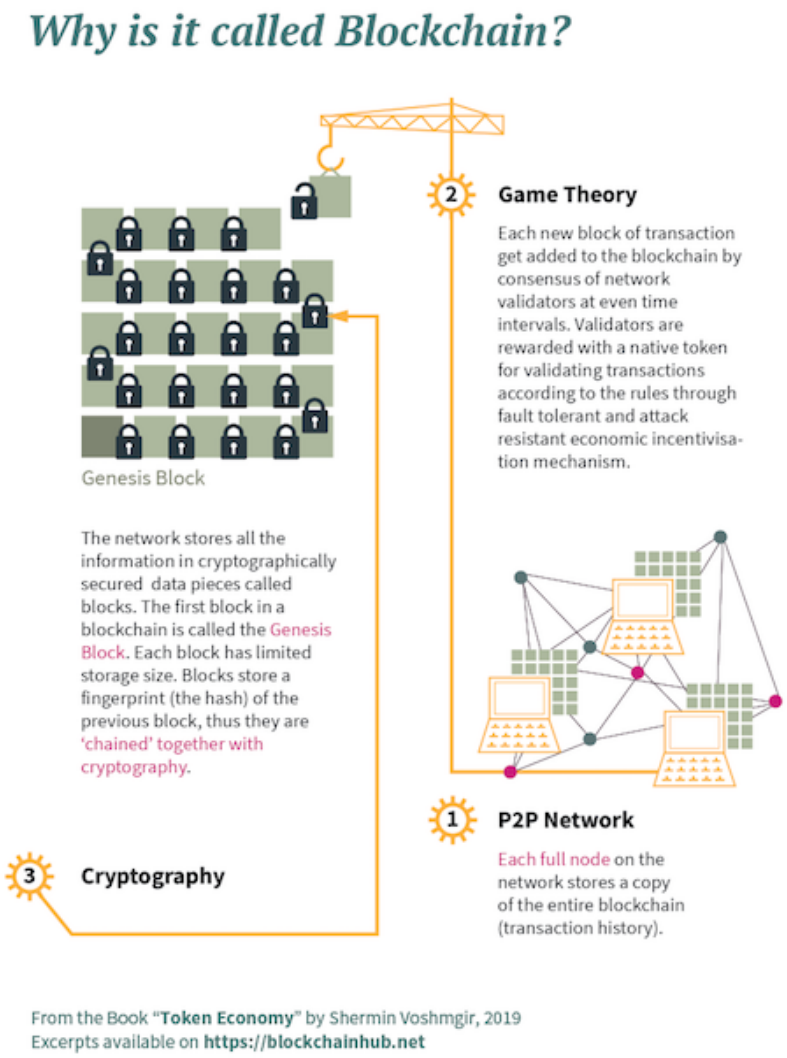

Figure 1.1: Why is it called Blockchain? (BlockchainHub 2022)

1.1.2 Addresses

Email addresses are used to determine where to send text or files, while Ethereum addresses determine where to send Ethereum transactions (Komodo 2022). Every account is defined by a pair of keys, a private key and a public key. Accounts are indexed by their address which is derived from the public key by taking the hash of the last 20 bytes (Ethereum 2022*b*). A private key is a 256-bit number which is like a username and password to an account. A private key generates its corresponding public key, and the address is a human-readable format of the public key. Using cryptography, the private key is used to sign transactions, which can verify that a user owns a private key to a particular public key without actually exposing the user's private key (BlockchainHub 2022)

1.1.3 Wallets

A blockchain wallet is a digital wallet that allows users to interact with the blockchain by controlling the use of the public key and a private key. An example of a wallet provider would be Metamask (Metamask 2022*a*) (See figure 1.2). A user can import their blockchain account by entering their private key into the Metamask app, stored locally on the device. The wallet allows users to view their cryptocurrencies and send transactions.

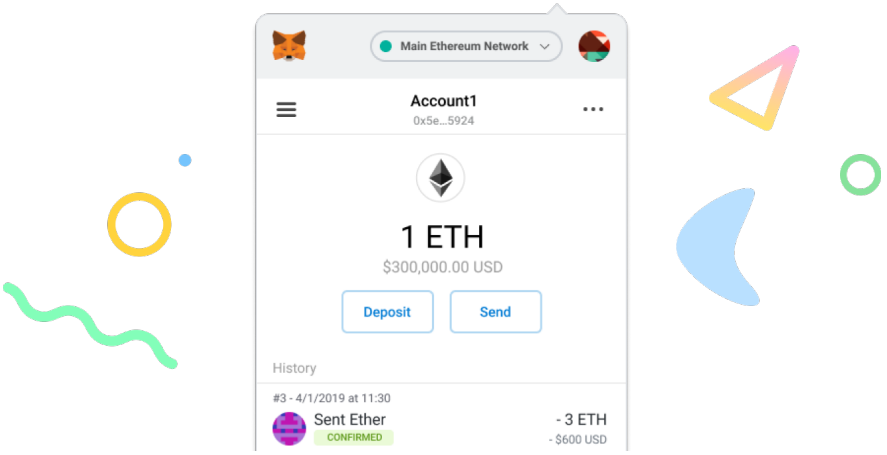

Figure 1.2: Metamask

1.1.4 Non Fungible Tokens

NFTs are a technology built on a blockchain that can prove ownership of digital assets such as collectables, artwork, in-game items and, in this case, event or concert tickets. Fungible means something interchangeable and will hold the same value when exchanged for something within its asset class, like gold, dollars, Bitcoin. For example, trading one bitcoin for another has the same total value. Non-Fungible is an asset that has unique attributes that make it different from other assets in its asset class like a painting, theatre ticket or a house. A theatre ticket would be non-fungible as trading it for another ticket will not be the same as it represents a different seat and may have a different value even though it is in the same asset class. A token is a digital certificate stored on a blockchain, which allows these digital assets to be publicly verifiable. As a result of NFTs utilising the blockchain, they inherit all the features of the blockchain of being verifiably original and more.

1.1.5 Smart Contracts

NFTs are built using Smart Contracts, which are programs stored on a blockchain that enable the execution of methods that read and write data on a blockchain. Smart contracts can be thought of as a backend API system. The program's current state at any moment in time is stored on-chain. For example, for event tickets, the smart contract would provide actions like allowing an event organiser to create new tickets, store what addresses own what ticket and allow the transferring of tickets. Also provided is the metadata for the ticket, which includes the ticket's name, image, and description.

Finally, any other business logic can be added, like specifying the maximum number of tickets. NFTs, adhere to the ERC-721 standard, which is essentially an interface that defines a set of rules for what methods or code must be included in the smart contract for it to be classified as a valid ERC-721 NFT. ERC-20 is another standard that is used to create tokens on the Ethereum blockchain like Chainlink (LINK), Tether (USDT) and Shiba Inu (SHIB). NFTs can be thought of like these tokens but having extra methods to attach information within the token. ERC-721 is helpful as it provides a standard for all NFTs, making it easier

for developers when creating NFTs as they know what they have to include in the code to make it an NFT. The standard also benefits developers integrating NFTs as they know what methods to call, allowing tools to be built for all NFTs using the ERC-721 standard like marketplaces.

1.2 Aims and Objectives

The main aims and objectives of this research are to:

1. Research current blockchain ticket solutions.
2. Explore the technology involved in an NFT ticketing solution and identifying improvements that can be made.
3. Evaluate and justify the best technology for NFT tickets.
4. Implement our own NFT ticket solution with identified improvements.
5. Create an app as a proof of concept demonstration of the proposed solution.
6. Evaluate the NFT ticket solution .

1.3 Methodology

To evaluate the use of NFTs as event tickets, we will first research current work done on NFT and blockchain ticketing, develop a design and then implement our own ticketing solution. We will then evaluate the solution by comparing the different characteristics involved in ticketing technology. We will also create an in-depth walkthrough of using our solution compared to current solutions to make a further evaluation, and finally discuss improvements and future work to be done with the solution.

2

Research

2.1 Background Research

The following research is information I have found where people discuss a company or event using blockchain tickets however in some cases it is not clear if what was described was implemented as a solution or if the implementation was successful. The implementation is also not open, in constrast to this paper, which hopes to provide an insight for future researchers in the area.

2.2 Previous Examples of Blockchain Tickets

2.2.1 2018 FIFA World Cup

Research shows that blockchain tickets were around back in 2018 (Dailyhodl 2018). The article describes that a company named BlocSide Sports, using the Aventus Protocol, was testing blockchain tickets for the 2018 World Cup. Whereas another article discusses that another company was testing a blockchain-based ticketing solution for the 2018 FIFA World Cup (Ethereum 2022*c*). This article describes that they are issuing tickets on the Ethereum blockchain using ERC-875, which is a token that supports batching and native atomic swaps, has since been withdrawn (Ethereum 2022*c*). A tweet (Twitter 2022) from Victor Zhang, who is affiliated with AlphaWallet describes that they implemented the FIFA World Cup 2018 NFT ticket provides screenshots within the tweet that

shows an app with an NFT ticket, but no other implementation details are released.

2.2.2 UEFA EURO 2020

UEFA for the 2020 euros, sold mobile tickets along with paper tickets on their blockchain-based mobile ticketing system (Dailyhodl 2018). This article describes that AlphaWallet and Victor Zhang, who was a part of the 2018 FIFA World Cup blockchain tickets, were also used for the UEFA blockchain tickets (UEFA 2020). However, Secutix, a ticketing company who has their app called TIXnGo published an article stating that they worked on the UEFA EURO 2020 ticketing solution and described details of what their partnership involved (Secutix 2022). The research is indeed confusing, and it is difficult to determine the validity of the research fully as the blockchain side of the tickets appear to be abstracted and mentioned briefly, and there is no way to track that these solutions were implemented since we do not have access to the blockchain information..

2.3 Companies making Blockchain Tickets

The following are companies that are involved in blockchain tickets and any information available regarding their implementation.

2.4 GET Protocol

GET protocol is an example of a company that does NFT tickets (GET 2022). They offer a white label solution for ticket providers to implement their NFT tickets for events. A company that utilises this technology is GUTS Tickets (GUTs 2022). GET protocol sells NFT tickets using the Polygon blockchain. They have provided an exciting website, where statistics and activity using the GET protocol can be viewed like tickets sold and recent tickets sold using their explorer (GET 2022). https://explorer.get-protocol.io/ This tool allows for viewing all the transactions that are being executed using GET NFT tickets. We can view these transactions on the PolygonScan blockchain explorer and visit the smart

contract address. GET protocol offers documentation, blogs, and the blockchain side of their code openly available.

2.5 Other Companies

Other companies that offer blockchain solutions are TIXnGO (TIXNGO 2022) which is made by Secutix, the company that was involved in the UEFA EURO 2020 tickets. Bam (BAM 2022) is another company that is first to appear when searching for NFT tickets. However, they only have a landing page showing what their ticketing solution involves, but no more information than that. Furthermore, Oveit (Oveit 2022) seems to provide a working solution for NFT tickets built on Polygon. Oveit is a company that already allows the sale of tickets online and to manage events with its event management system. However, they have integrated NFT tickets as a solution too.

2.6 Summary

The solution that would be most similiar is the solution provided by GET Protocol, however our solution will provide extensive detail on the technology and justification of choices. Furthermore, the solution is more catered for people, who are unfamiliar with blockchain technologies, and highlights how such a system can be created from design to implementation. By breaking down every step in the process, it can bring more awareness to the technology, and show how people can implement their own solutions.

3

Design

The goal of the design chapter is to create a minimal viable product for an NFT event ticketing system and to create a deeper understanding in what is involved in developing such a system. The solution aims to select the Web3 approach for all requirements and evaluate these choices in the evaluation chapter.

3.1 Requirements

The first step of the design process was to figure out the requirements for the solution. The requirements reveal the scope of the solution for a minimal, high-level implementation of NFT event tickets. Any NFT ticket solution, regardless of complexity, would be derived from the requirements as outlined. The requirements were drawn from personal experiences of attending events. The process of attending an event could be broken down into several requirements. For the requirements, a bouncer would refer to someone or something that validates the ticket at the venue and grants entry if the ticket is deemed valid—for example, a human bouncer or a turnstile gate.

3.1.1 Functional

- Users shall be allowed to purchase tickets.
- Users shall be allowed to trade their tickets with other users.

- Bouncers shall be allowed to validate that a user owns a ticket for a particular event.

3.1.1.1 User Stories

Based on the requirements, the user stories shown in figure 3.1 were created.

Purchase Tickets
As a user I want to purchase tickets So I can attend a particular event
Acceptance Criteria: Be able to purchase tickets through the smart contract

Trade Tickets
As a user, I want to trade tickets So I can be reimbursed for my purchase if I change my mind
Acceptance Criteria: Be able to trade tickets on a marketplace

Validate Tickets
As a bouncer, I want to validate tickets So I can ensure a person has a valid ticket
Acceptance Criteria: A third party can validate ownership of a ticket Tickets cannot be used more than once

Figure 3.1: User Stories

3.1.1.2 Use Case Diagram

The use case diagram was derived from the user stories, which helps in highlighting the actors involved in the solution and their responsibilities to aid in further visualising the scope of the solution.

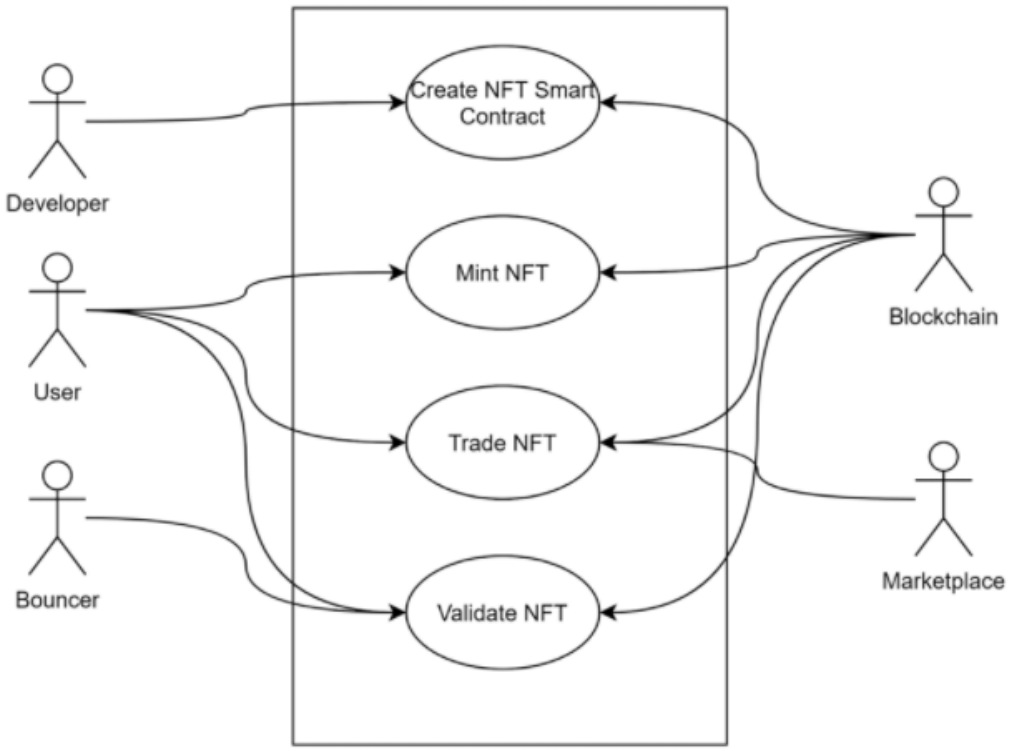

Figure 3.2: Use Case Diagram

3.1.2 Non-Functional

- The system shall be secure to prevent fraud and theft.
- The system shall be extensible so that new or modified code shall not affect existing code.
- The system shall be reliable to ensure all users can attend events without issues.
- The system should be performant in that throughput should not cause delays at venue doors.

3.2 Choice of Blockchain

The Ethereum blockchain popularised NFTs. The Ethereum blockchain has supported NFTs the longest and has extensive tooling and support for development. While other blockchains exist that support NFTs, Ethereum has been chosen as the solution for its ecosystem of resources for blockchain development, and Ethereum fulfils the requirements. One issue of Ethereum is its slow throughput. Transactions are stored in groups named blocks. The network validates these blocks to ensure the validity of the transactions. The time it takes to validate these blocks represents how long it takes for a transaction to be published on the blockchain. Between January 1-7, 2022, the average block time on Ethereum was 13.2 seconds, with roughly 188 transactions per block. On Polygon, the average block time during the same period was 2.3 seconds, with 48 transactions on average per block. This means, in the same 13.2 seconds, Polygon was able to confirm on average 271 transactions—44% more transactions than the Ethereum network (Blocknative 2022). Polygon is a "layer two" or "sidechain" scaling solution that runs alongside the Ethereum blockchain — allowing for speedy transactions and low fees (Coinbase 2022). Purchasing the ticket, trading the ticket and validating the ticket are all transactions that need to be recorded on the blockchain with the current solution. Polygon inherits all the features of Ethereum, except they use their own scaling solution. Their solution is still involving the Ethereum ecosystem; the same tools, smart contracts and resources can be used. 2.3 seconds versus 13.2 makes Polygon usable in our solution, as the latter is not feasible for real-world use.

3.3 Proposed Solution

The functional requirements are stated as being able to purchase, trade and validate tickets. The requirements outlined cover all the use cases. The goal of the proposed solution is to produce a minimal viable product for NFT tickets to show that the solution works and then create a dApp to show what is involved in connecting an application to a smart contract and improve the user experience. This section will describe and justify the design choices made. The overall solution

features a decentralised approach where users can rely on existing tools to interact with the application. This approach allows for a working implementation in a quicker timeframe to prove the solution is functional. For the minimal viable product, Polygonscan (cite polygonscan) is used as the user interface, which is a blockchain explorer for Polygonscan. The blockchain explorer allows us to view the smart contract, and interact with the methods using our Metamask mobile crypto wallet, to purchase, trade, or verify tickets.

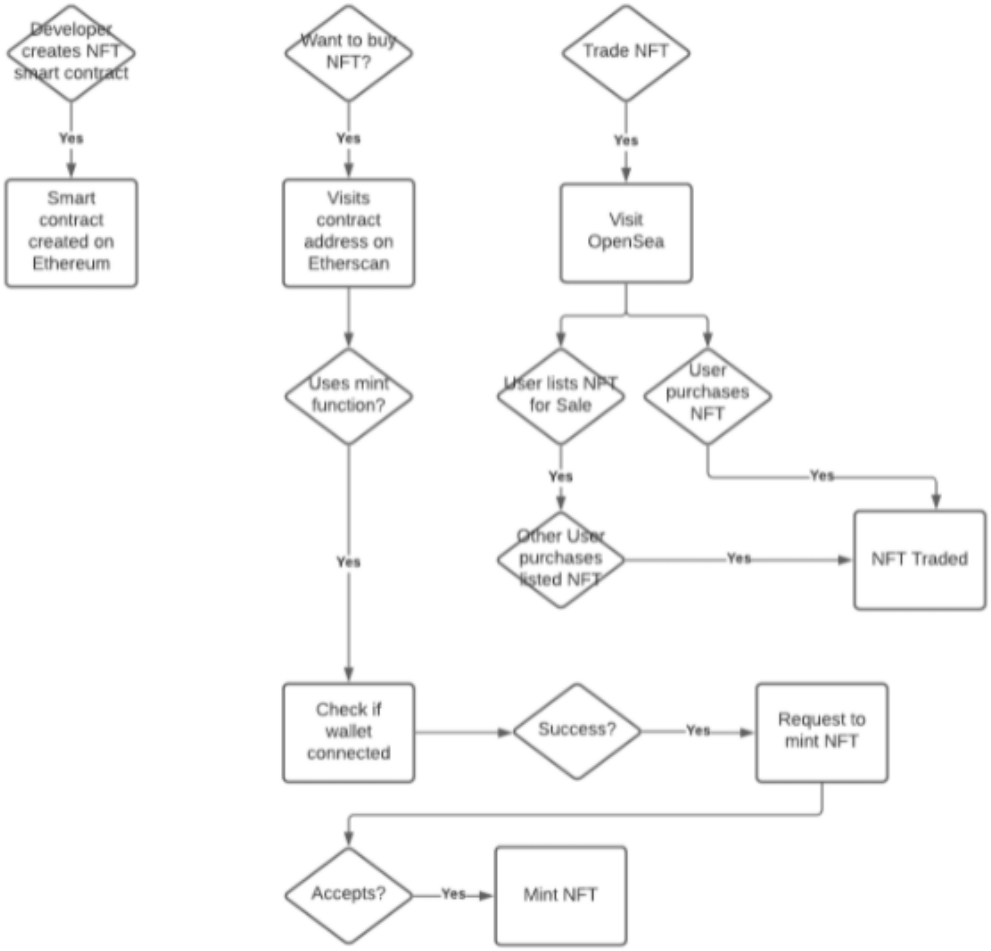

Figure 3.3: Flow chart of the Purchasing and Trading Requirements

- **Purchasing Tickets:** The ability to purchase tickets is included by creating the smart contract for the NFT tickets. The ERC-721 standard includes a method called `safeMint` which is used to create a new ticket NFT. Minting means creating the NFT on the blockchain. This method can be designed to require a price to be paid to mint the NFT, therefore making the NFT ticket purchasable.

- **Trading Tickets:** ERC-721 also provides support for trading NFT tickets. ERC-721 NFTs include the `safeTransfer` method and `setApprovalForAll` method. The `safeTransferFrom` method allows users to trade tickets peer

to peer by specifying an address they want to transfer their NFT ticket to. Since NFTs adhere to the ERC-721 token standard, third-party developers can quickly implement NFTs into their solutions as a solution for one ERC-721 compliant NFT works for all NFTs. OpenSea is an example of a marketplace that allows for the trading of NFTs. OpenSea is the most popular marketplace for buying, selling and trading NFTs. They periodically scan the blockchain for all NFTs, keep track of what wallets own what NFTs, and facilitate the trading of any NFT on their marketplace website. Once our smart contract is created, OpenSea will detect it and display it on their website. Users can list their NFTs for sale and purchase other NFT tickets on OpenSea. The marketplace works by the user interacting with OpenSea's marketplace contract by using the `setApprovalForAll` method, which allows the smart contract to transfer the NFT on the user's behalf given the agreed sale price is met. Other marketplaces exist like LooksRare and Rarible; however, OpenSea has support for Polygon NFTs and Polygon test networks so we can view our NFTs in development.

- **Verifying Tickets:** The final user story to fulfil and the most complex is validating the NFT ticket. When coming up with the design of the ticket validation, an issue was making sure that the person standing directly in front of the bouncer was the right person with the NFT ticket. This is a core problem and without this, the ticket solution would fail. Two-factor authentication was implemented into the solution to improve the non-functional requirement of making the solution secure.

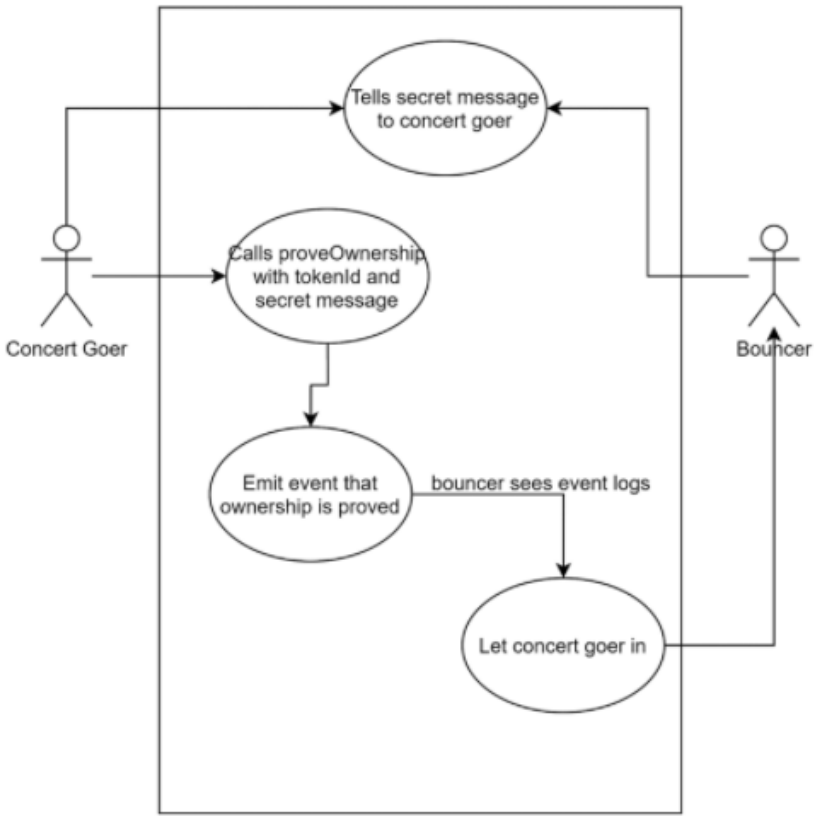

Figure 3.4: Use Case Diagram

Figure 3.4 illustrates the solution for the validation of the NFT ticket. The design was formed by walking through a real-life scenario. Imagine a concert goer arrives at a bouncer and tries to enter the venue of an event they have an NFT ticket for. The bouncer tells a secret code only known to the bouncer and the concert goer, for example, '3459'. The secret code is used as a form of two-factor authentication to make the verifying of ownership more secure, to know that the concert-goer that executed the contract method to prove ownership is not an external person. If the secret code resolution were omitted, the solution would not be able to prove attendance. The concert-goer uses their mobile wallet, for example, Metamask (Metamask 2022*a*), a mobile wallet that provides a user interface to utilise a wallet and interact

with the smart contract. To interact with the smart contract, concert-goers and bouncers can use a blockchain explorer like Etherscan (Etherscan 2022), allowing navigation to view all details about the smart contract and execute methods. They then call the `proveOwnership` method in the smart contract with their ticket ID and the secret code '3459' as arguments. The ticket ID would be the token ID, which ERC-721 assigns to each NFT minted. So the first NFT minted would be token ID 0, and the second NFT minted would be token ID 1, and so on. If the concert-goer owns the specified NFT ticket ID, then the smart contract method will emit an event on the blockchain displaying that they have proved ownership for the respective ticket ID and the secret code. The bouncer can view the smart contract's event logs at the smart contract address on the blockchain explorer and see the event log to verify that the secret code matches the specified secret code, proving that they own the NFT ticket. They can then let the concert-goer into the venue.

3.4 Metadata

NFT metadata describes the information about the NFT, such as its name, description and image, that marketplaces like OpenSea use to display NFTs. The metadata standard is defined in the ERC-721 interface with what needs to be included in the JSON. The tokenURI method of the smart contracts returns the metadata for a particular NFT that the smart contract creator has to set. For the metadata for this project, base64 encoding was decided as a solution to encode the details of the NFT as it allows the metadata to be on-chain and exist forever. However, for the NFT image, the file size is too large to store on-chain as the transaction fees would be too high. The solution for image storage is a distributed file store like IPFS or Filecoin. For this solution, IPFS was decided, and a service called Pinata was used to upload the image for ease of use. It is not essential for the solution, but it is necessary to highlight what is involved in creating NFTs, which is why the design choices for the Metadata are not substantial.

3.5 Dapp Solution

The goal of the dApp is to abstract the functionality of the smart contract, making the dApp easier to use for both the bouncer and end-user while showcasing the steps involved in creating a dApp that interacts with the blockchain. The dApp solution will include a user interface to mint an NFT, validate an NFT and show event logs for the bouncer to view the validations in a more readable and accessible manner.

3.6 Development Environment

For a development environment, Hardhat.js (Hardhat 2022*a*) was decided. During learning, Remix IDE, an online IDE for Ethereum was used to create a smart contract. Issues were verifying the smart contract on the blockchain explorer Etherscan. Without the smart contract being verified, the source code cannot be viewed, and methods like proveOwnership cannot be executed as the byte code is not decompiled and readable. Hardhat and Truffle were two other solutions for accomplishing this, and people seemed to prefer Hardhat and noted their frustrations with Truffle (abcoathup 2022). The Hardhat tutorial was comprehensible and quick to set up, securing Hardhat as the choice for the development environment.

4

Implementation

The implementation chapter goes through all the steps involved in creating the smart contract for the NFT ticket and the dApp from start to finish.

4.1 Smart Contract Setup

4.1.1 Hardhat.js

To set up hardhat.js, Node.js has to be installed if not already. This installation can be done through an installer (Nodejs 2022).

The following commands were run in the terminal to create the project folder.

1. mkdir NFT-Event-Tickets.
2. cd NFT-Event-Tickets.
3. npm init –yes
4. npm install –save-dev hardhat
5. npx hardhat

The Create an empty hardhat.config.js was selected from the options. For our code and tests, two new folders were created named “contracts” and “test” under the current directory. Next, the solidity visual studio code extension needed to be

installed (Blanco 2022). Once installed, a new folder named .vscode was created with a file named settings.json with the contents shown in Figure 4.1

Figure 4.1: Vscode Settings

These settings help prevent import and compile errors later. The development environment is now set up at this stage. Some other configurations will be implemented as they are met, like compiling, testing, and deploying the smart contract.

4.1.1.1 Setting up a Blockchain Wallet and Address

An address and a wallet provider are needed to interact with the blockchain. For the implementation, Metamask is used to support the creation of addresses and offer a mobile wallet to use at events. Metamask is a browser extension that can be download by navigating to their website and clicking the download button (Metamask 2022*b*). Once installed, the extension will appear on the browser, and clicking the extension will begin the guided setup process by Metamask.

1. Click Get Started.
2. Create a Wallet.
3. Agree or refuse to help improve Metamask.
4. Create a password.
5. Write down the secret recovery phrase.
6. Confirm the secret recovery phrase.

The password created is used locally to prevent people who have access to the same computer from using Metamask. The secret recovery phrase is a 12-word phrase that is the "master key" to your wallet and your funds (Metamask 2022*a*). The secret recovery phrase is the private key in word format. Metamask uses BIP39, which uses a mnemonic phrase to serve as a backup to recover the private key. The private key can be derived from this phrase (BlockchainHub 2022).

4.1.2

Connecting to the Test Network Since Ethereum is a protocol, there can be multiple independent "networks conforming to this protocol that do not interact with each other. Networks are different Ethereum environments that can be accessed for development, testing, or production use cases. (Ethereum 2022*d*) The Mainnet is the primary Ethereum blockchain used in production. This network is where real value transactions occur, and real money is paid to interact with the blockchain. We do not want to pay real money to develop our applications; that is why Testnets also exist. These networks provide an environment identical to Mainnet, except there is no value to any transactions. Most Testnets use a proof-of-authority consensus mechanism. A small number of nodes are chosen to validate transactions and create new blocks. Testnet tokens are needed to pay the transaction fees. These tokens can be gotten from a faucet, which are dApps that send Testnet tokens to a specified address to request Testnet tokens. The Polygon test network Mumbai is being used for the implementation, which is the main Testnet for Polygon. The Testnet can be connected to using Metamask. The currently connected network can be viewed at the top header from the browser extension. If it is the first time using Metamask, the Ethereum Mainnet will be displayed as the connected network.

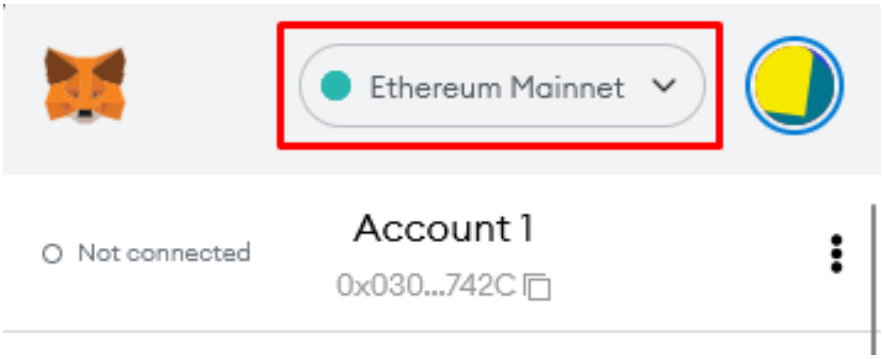

Figure 4.2: Metamask connected network

Clicking on the dropdown displays a list of networks that the wallet can switch to.

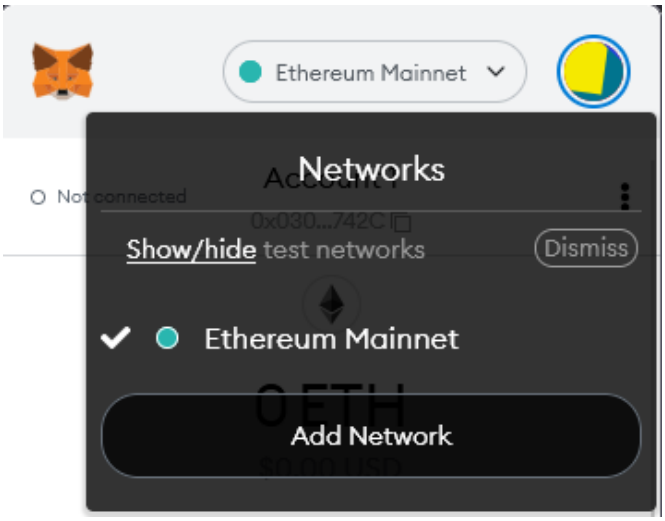

Figure 4.3: Metamask available networks list

The Mumbai Testnet must be added as a network by clicking the Add Network button, which redirects to a user interface where the network details can be entered. The official Polygon documentation shows the network details (Poylgon 2022). This information was used to enter the network details.

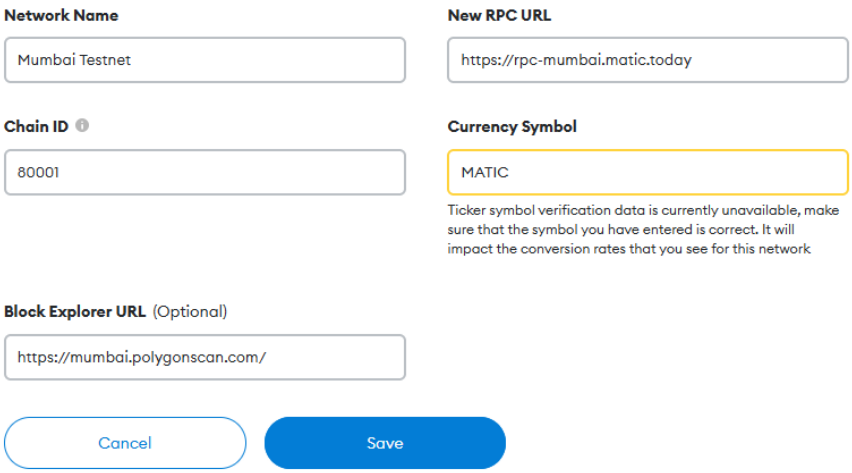

Figure 4.4: Metamask adding network

The save button is used to save the network. The network can be selected from the dropdown list, and the wallet can be connected to the Mumbai Testnet.

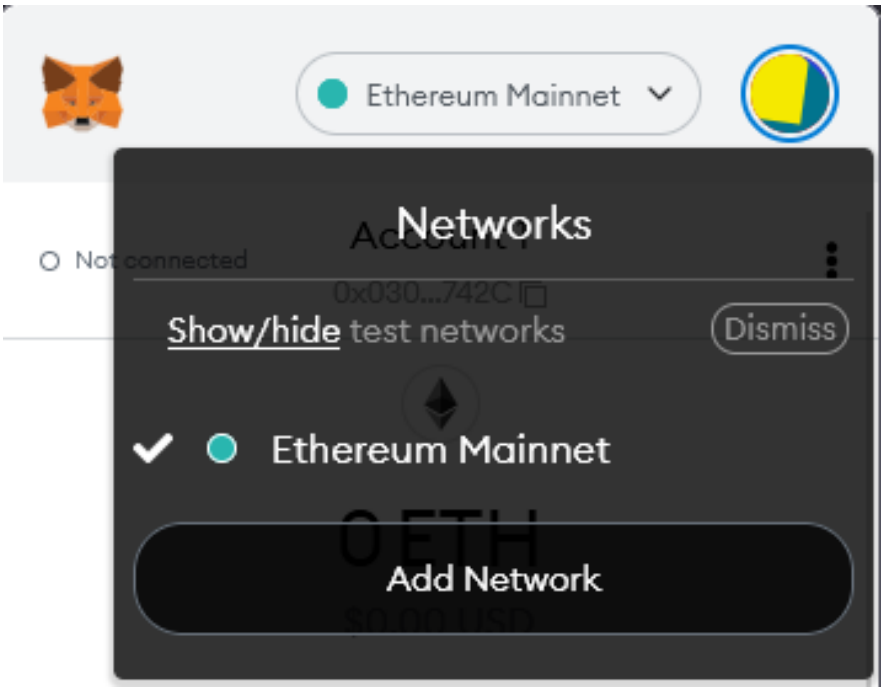

Figure 4.5: Metamask available networks list with Mumbai Testnet

4.2 Smart Contract Setup

4.2.1 Smart Contracts Wizard

The OpenZeppelin contracts wizard was used to generate a starter ERC-721 template. OpenZeppelin has built smart contract templates that are audited, secured and community-vetted ERC compliant smart contracts for developers to use. The wizard allows for customising the smart contract to what is needed. Using OpenZeppelin saves time, and there is increased reliability and security as they adhere to the best development practices. The following features were selected in the contract wizard UI for the solution: "Mintable", Auto Increment Ids", "Enumerable", and "URI Storage" to build the smart contracts. The wizard then presents a fully functional ERC-721 NFT smart contract to build upon. (OpenZeppelin 2022)

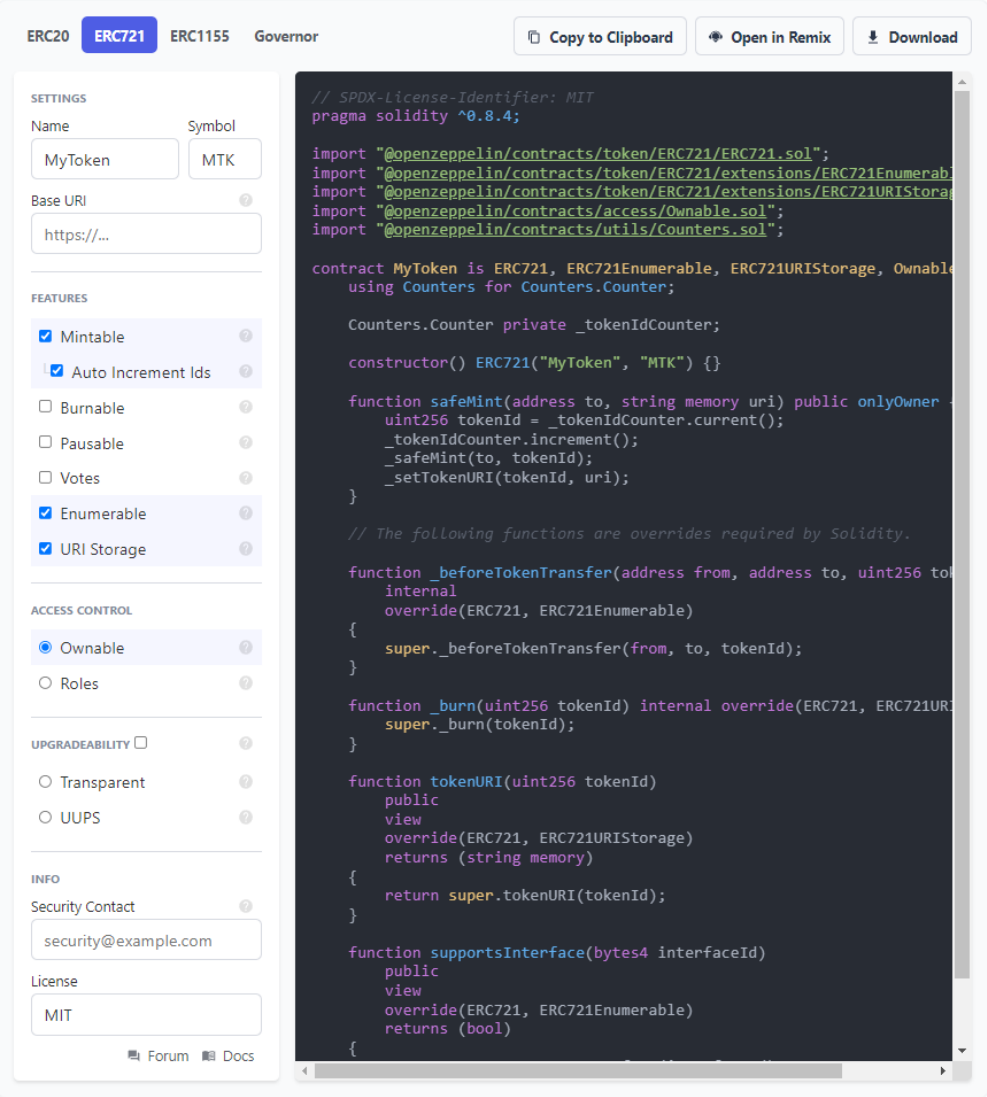

Figure 4.6: OpenZeppelin Smart Contracts Wizard

The file MyToken.sol is created under the contracts folder in our hardhat project, and the code from the Smart Contracts Wizard is added to this file to use this code. The sol file extensions are used for Solidity programs. An error was given for the OpenZeppelin imports starting at line 4 in the code. To fix this, OpenZeppelin was installed using the npm package manager. The compiler version in the config file also needed to be changed to 0.8.4 in the hardhat.config.js file to match the solidity version in the smart contract file. The command npx

hardhat compile could now be run to compile the code successfully. The smart contract now supports the minting and trading of NFTs. However, it is still missing the Metadata and verifying ownership functionality.

4.2.2 Verifying Ownership

To implement verifying ownership, the code shown in Figure 4.7 was written for the implementation.

```
function proveOwnership(uint256 tokenId, uint16 secret) external {
    require(msg.sender == ownerOf(tokenId));
    emit OwnershipApprovalRequest(msg.sender, tokenId, secret);
}
```

Figure 4.7: proveOwnership

The method name is defined proveOwnership with the parameters tokenId and the secret. The keywords uint followed by the number, for example uint256, is an unsigned integer, and the number defines how many bits the variable can store. The secret parameter is of type uint16 as the secret code is only a four digit number, so it does not need to be a larger type, as 16 bits can hold numbers up to 65,535. The external keyword means the function can only be used outside the contract. The require keyword in line two checks if a condition is true, and if it is not, it will stop the execution of the method. This line of code checks the msg.sender, the address who called the method, is owner of the tokenId, which is the id of the ticket that is trying to be verified. The ownerOf method is implemented in the ERC721 contract and returns the address of the owner of a particular tokenId. This line of code proves ownership of the NFT ticket. The OwnershipApprovalRequest event is emitted with the address, tokenId and secret as arguments, which announces the event on the blockchain for everyone to view.

The event first has to be defined before it can be emitted, as shown in Figure 4.8.

```
event OwnershipApprovalRequest(address ownerAddress, uint256 tokenId, uint16 secret);
```

Figure 4.8: OwnershipApprovalRequest

The smart contract now implements verifying the NFT ticket successfully.

4.2.3 Metadata

NFT metadata defines the NFT as an object containing details about the digital asset. Metadata stores information such as the name, description, attributes and image of the NFT. The metadata for an NFT is returned by the tokenURI method. EIP-721 (Ethereum 2022*e*) defines metadata standards. The metadata standard is a template for what we can include in our metadata. Marketplaces like OpenSea use this metadata to pull data like the name and the image of an NFT.

For the NFT image, an artwork was created and uploaded to IPFS using Pinata, a service for hosting NFT media. The image was uploaded to Pinata and they provided a unique ID representing the IPFS hash. The image is now uploaded to IPFS and can be retrieved using that ID (IPFS 2022).

The next step was to update the tokenURI method to return the metadata associated with the NFT. The base64 solidity library is needed to encode the metadata, so it takes up less memory on the blockchain, making the smart contract cheaper. This library is installed via the npm package manager (Brechtpd 2022).

The library is then imported along with the Strings library from OpenZeppelin, to convert any uint to a string.

```
import "@openzeppelin/contracts/utils/Strings.sol";
import "base64-sol/base64.sol";
```

Figure 4.9: imports

The tokenURI implementation is shown in 4.10..

```
string imageURI = "ipfs://QmfZ6txFKSi7ukredN7c8dCPaqejAE3bY2VnNxnLnqLhA9";

function tokenURI(uint256 tokenId)
    public
    view
    override(ERC721, ERC721URIStorage)
    returns (string memory)
{
    string memory tokenIdToString = Strings.toString(tokenId);
    string memory json = Base64.encode(
        bytes(
            string(
                abi.encodePacked(
                    '{"name": "NFT Ticket #', tokenIdToString,
                    '", "description":
                    "An NFT based version of traditional Event Tickets",
                    "image": "', imageURI, '"}'
                )
            )
        )
    );
    return string(abi.encodePacked("data:application/json;base64,", json));
}
```

Figure 4.10: tokenURI method

The imageURI links the image we want to use for our Metadata. The name of each ticket is made up of the tokenId, which is why we need the Strings library to convert to tokenId to a string. The base64 library encodes our Metadata, so it uses less memory on the blockchain. Services like OpenSea use this Metadata to display details regarding the NFT.

4.2.4 Testing the Smart Contract

Smart contracts are immutable once deployed, which means that once deployed, we cannot change any of the code. Immutability is great for verifiability but also means that any bugs encountered after deployment cannot be dealt with. Immutability is why testing is essential to smart contract development, as any minor error can be catastrophic and put smart contract users at risk. A new file was created under the test folder named MyToken.js to begin writing tesst. The

tutorial on the hardhat page was used to construct the tests (Hardhat 2022*b*). The goal of the testing was to show how tests can be written and make sure all the requirements worked.

```
const { expect } = require("chai");

describe("Token contract", function() {
    let Token;
    let hardhatToken;
    let owner;
    let addr1;
    let addr2;
    let addrs;

    // `beforeEach` will run before each test, re-deploying the contract every
    // time. It receives a callback, which can be async.
    beforeEach(async function() {
        Token = await ethers.getContractFactory("MyToken");
        [owner, addr1, addr2, ...addrs] = await ethers.getSigners();

        hardhatToken = await Token.deploy();
    });

    describe("Deployment", function() {
            it("Should set the right owner", async function() {
                expect(await hardhatToken.owner()).to.equal(owner.address);
            });
     });
});
```

Figure 4.11: Test code

The variables that are used are defined at the top. The token is deployed, and we define signers, which are addresses we want to use for testing, the first address being the owner of the contract. The contract is deployed in the simulated blockchain environment, and the tests we describe can then be run.

- **Deployment:** The first test was taken from the hardhat tutorial and checks if the owner was set correctly. This test case is vital because if the incorrect owner is set or is not set at all, it voids any smart contract methods that the contract owner can only call.
- **Mint an NFT:** For the to mint an NFT test, the number of event ticket NFTs an address has is checked, also known as the token balance for the

address. The safeMint method is then called, and the token balance is rechecked after the method is called. If the balance has increased by one, the mint was successful as the address has one additional token.

- **Transferring an NFT:** For the transferring NFT test case, a similar structure was followed. An NFT is minted to address one so that they have an NFT to transfer. If minted successfully, the token balance of address one and address two is recorded. The transferFrom method is used to transfer token Id 0 to address two. Checks are then done to check if the owner of token Id 0 is address two and if the token balance of address one has decremented and address two has incremented, signifying that the NFT has been successfully transferred.

- **Verifying an NFT:** For the validation test case, minting an NFT to the wallet is done. A secret code is then defined, and the proveOwnership method is called with tokenId 0 and the secret code as arguments. The event OwnershipApprovalRequest is then checked to see if it emitted with the arguments of the address, tokenId and secret.

The tests can be run using the npx hardhat test command, and the terminal will display if the tests passed or failed and how long they took to run. Implementing these test cases helped identify issues while coding the smart contract and helped significantly with future versions. If any changes are made, the tests will run to ensure it does not break any current code features.

4.2.5 Deploying the Smart Contract

A new folder called Scripts was created to deploy the contract, and a new file under Scripts called deploy.js was created. The code from the hardhat tutorial website was taken for deploying the contract (Hardhat 2022*b*). The only change was changing Token to MyToken

```
async function main() {
    const [deployer] = await ethers.getSigners();

    console.log("Deploying contracts with the account:", deployer.address);

    console.log("Account balance:", (await deployer.getBalance()).toString());

    const Token = await ethers.getContractFactory("MyToken");
    const token = await Token.deploy();

    console.log("Token address:", token.address);
}

main()
    .then(() => process.exit(0))
    .catch((error) => {
        console.error(error);
        process.exit(1);
});
```

Figure 4.12: Deploy code

To deploy the contract to the Mumbai Testnet, changed needed to be made to the hardhat.config.js file to support this. The requirements for this file are to get an API key from an Ethereum provider to interact with the blockchain, specify the network, and specify our private key so that the deployer can use that account. For a provider, Alchemy was used. They offer a free tier and support connecting to Polygon's Mumbai Testnet. Providers are services that run nodes that talk to the Ethereum network. They abstract interaction with the blockchain by providing API endpoints for developers to use, which means we do not have to run our nodes to connect to the blockchain. Another provider would be Infura, which is what Metamask uses. The first step is to create an Alchemy account to get an API key. After logging in from the home screen, the create app button triggers a form to fill out details about the app. Polygon was selectable as a chain, and Polygon Mumbai as the network. An app was then created, which allowed viewing and copying the Alchemy API Key needed for the config file. The API key and the private account key were added to the config file.

```
require("@nomiclabs/hardhat-waffle");
require("@nomiclabs/hardhat-etherscan");
require("dotenv").config()

const ALCHEMY_API_KEY = process.env.ALCHEMY_API_KEY;
const MUMBAI_PRIVATE_KEY = process.env.MUMBAI_PRIVATE_KEY;
const POLYGONSCAN_API_KEY = process.env.POLYGONSCAN_API_KEY;

/**
 * @type import('hardhat/config').HardhatUserConfig
 */
module.exports = {
    solidity: "0.8.4",
    networks: {
        mumbai: {
            url: `https://polygon-mumbai.g.alchemy.com/v2/${ALCHEMY_API_KEY}`,
            accounts: [`0x${MUMBAI_PRIVATE_KEY}`],
        },
    },
    etherscan: {
        apiKey: POLYGONSCAN_API_KEY
    }
};
```

Figure 4.13: hardhat.config.js file

The deploy command can now be executed 'npx hardhat run scripts/deploy.js –network mumbai'. This command runs the config file's deploy script for the specified network. After the command finishes executing, we are given the smart contract address, where the code is deployed on the blockchain. The smart contract address is '0xcaa7cfdd22c401db2addae7dc7a7cbd7fb84b260' We can view the smart contract using the Mumbai PolygonScan blockchain explorer (PolygonScan 2022).

4.3 Dapp Implementation

4.3.1 Setting up the Development Environment

Next.js is a server-sided React Framework for developing applications. Next.js is a personal preference for the frontend. Majority of frontend technologies can be used to create the dApp. To setup Next.js I followed the tutorial on the

nextjs website (NextJS 2022). For the Web3 library, ethers.js (EthersJS 2022) was used to interact with the Ethereum blockchain and ecosystem. To aid with the front-end design, Mantine UI, a react component library was used.

4.3.2 Authenticating the User

The first step of the dApp is authenticating the user. A wallet is used in Web3 to sign into websites. This functionality is implemented into the Sign In button. When the Sign In button is clicked, it will prompt the user to connect their wallet address. There is a javascript library called Web3Modal, which provides an API to allow us to support multiple wallet providers, not just Metamask.

```
const providerOptions = {
    metamask: {
        package: true
    },
    walletlink: {
      package: WalletLink,
      options: {
        appName: "Web 3 Modal Demo",
        infuraId: "12345678assacagfdg24525afsaftrs3",
        rpc: "",
        chainId: 4
      }
    },
    walletconnect: {
      package: WalletConnect,
      options: {
        infuraId: "12345678assacagfdg24525afsaftrs3"
      }
    }
};
```

Figure 4.14: providerOptions

```
const connectWeb3Modal = async () => {
  try {
      const web3Modal = new Web3Modal({
          cacheProvider: true, // optional
          disableInjectedProvider: false,
          providerOptions // required
      })

      const provider: any = await web3Modal.connect()
      const library = new ethers.providers.Web3Provider(provider)
      setProvider(provider)
      updateUser(library)
  } catch (error) {
      console.error(error);
  }
};

const updateUser = async (provider: ethers.providers.Web3Provider) => {
    const signer = provider.getSigner()
    let address = await signer.getAddress()
    let ens = await resolveEns(provider, address);

    let displayName;
    if (ens) {
        displayName = ens;
    } else {
        displayName = address.substring(0,6) + "..." +
                      address.substring(address.length-4, address.length);
    }

    let network = await provider.getNetwork()

    setUser({
        address: address,
        displayName: displayName,
        network: network.name
    })
}
```

Figure 4.15: Authentication code

The providerOptions variable is needed to allow the other wallet providers to work. An Infura API key is needed for it to work, which can be retrieved from the Infura website (Infura 2022). The infuraId in the code is random, and not the actual API key used. Providers are an abstraction of a connection to the Ethereum network, whereas signers are an abstraction of an Ethereum account.

The connectWeb3Modal method is called when the Sign In button is clicked. This button will open the modal with provider options, and if the user connects successfully, it will update our user and provider state. The method retrieves the address of the connected network and uses the ENS name as a display name instead of the address if it exists. This method is also run on every page load using the useEffect hook. If the account is already connected to the website, it will save the data to state without asking the user to connect. This check is needed to see if the account is connected when navigating pages, so the state persists across pages.

4.3.3 Handling Changes

The user can switch addresses, disconnect their wallet or switch networks. These are situations that have to be handled. Event listeners provided by Web3Modal are used. These event listeners give the user full transparency and feedback on how the dApp reacts to their changes and whether the dApp can continue in that state. For example, if the user is on or switches to a network that is not the Mumbai Testnet, the dapp must notify that the user has to change network for the dApp to continue to work.

```
useEffect(() => {
  if (provider?.on) {
      const handleAccountsChanged = (accounts: any) => {
          console.log("accountsChanged", accounts);
          const lib: any = new ethers.providers.Web3Provider(window.ethereum, "any");
          if (accounts) updateUser(lib);
      }

      const handleChainChanged = async (chainIdHex: string) => {
          const lib = new ethers.providers.Web3Provider(window.ethereum, "any");
          updateUser(lib)
      }

      const handleDisconnect = () => {
          disconnect();
      }

      provider.on("accountsChanged", handleAccountsChanged);
      provider.on("chainChanged", handleChainChanged);
      provider.on("disconnect", handleDisconnect);

      return () => {
          if (provider.removeListener) {
            provider.removeListener("accountsChanged", handleAccountsChanged);
            provider.removeListener("chainChanged", handleChainChanged);
            provider.removeListener("disconnect", handleDisconnect);
          }
      };
  }
}, [provider])
```

Figure 4.16: Handling Changes code

This useEffect hook updates when the provider state changes. Event listeners will be added if the provider exists, and will be removed when the provider changes. The code handles when a user switches their account, switches network or disconnects, and updates the state for the frontend to react to the changes.

4.3.4 Purchasing a Ticket

The following implementation of the dApp was to abstract purchasing of the ticket. For this requirement, A button called purchase ticket was created, and when clicked, it will call the safeMint function from our smart contract, which will prompt a transaction for the connected user's wallet. If the user confirms

the transaction, then they have purchased the ticket. This experience is similar to executing the safeMint write method from the smart contract address on the blockchain explorer. The smart contract ABI and the smart contract address is needed to implement the code. ABI stands for Application Binary Interface, the standard way to interact with contracts in the Ethereum ecosystem (Solidity 2022). The ABI is gotten from the smart contract, and we only need to specify the methods we need.

```
const abi = [
  "function tokenURI(uint256 _tokenId) external view returns (string)",
  "function ownerOf(uint256 _tokenId) external view returns (address)",
  "function safeMint(address to)",
]
```

Figure 4.17: ABI

Now the code for the contract can be written.

```
const contractAddress = "0xf0d755b10b0b1b5c96d00d84152385f9fd140739"

const purchaseNFT = async () => {
    const provider = new ethers.providers.Web3Provider(window.ethereum);
    const signer = provider.getSigner();

    const contract = new ethers.Contract(
        contractAddress,
        abi,
        signer
    );

    try {
      notifications.showNotification({
        title: 'Transaction Started',
        message: 'Confirm the transaction to continue',
      })

      const verifyTxn = await contract.safeMint(user.address)

      const id = notifications.showNotification({
        title: 'Transaction Sent',
        message: 'The transaction should be confirmed shortly',
      })

      await verifyTxn.wait()

      setTimeout(() => {
        notifications.updateNotification(id, {
          id,
          color: 'teal',
          title: 'Transaction Confirmed!',
          message:
            'Notification will close in 2 seconds, you can close this notification now',
          icon: <CheckIcon />,
          autoClose: 2000,
        });
      }, 3000);
    } catch (error) {
      notifications.showNotification({
        title: 'Error',
        message: error.message,
        color: 'red'
      })
    }
```

Figure 4.18: purchaseNFT

The Mantine notification system is used here to provide real-time updates to

the user about the transaction status. First, we get the user and initialise the contract. The safeMint method from the smart contract is called, and the notification system display that the transaction has started. This method is a promise, so if the user confirms the transaction, the notification system will display that the transaction has been sent and should confirm shortly. If the user rejects the transaction or any other error occurs in the process, the notification system will catch the error and display the error message. The code await verifyTxn.wait() is used to wait until the transaction confirms. Once the transaction confirms the notification system displays that it has confirmed, which means the ticket has been purchased.

4.3.5 Proving Ownership of a Ticket

The Scan Tickets button is created for implementing the ownership of the ticket. This button then brings the user to another page displaying all of the user's event ticket NFTs. Alchemy is used to display the NFTs for the user, which is a blockchain API service that has endpoints tailored for NFTs to make retrieving NFTs from the blockchain easier. Using the user's address that is gotten from them connecting their wallet, a GET request is sent to the getNFTs endpoint using the Alchemy API key, the user's address, and the smart contract address for the NFT ticket. This endpoint will return NFTs owned by the user. The Metadata is then needed for each NFT. The IPFS image needed the IPFS URL appended at the start. Websites like OpenSea automatically recognise the IPFS hash and read it. The way it is implemented in this solution does not appear to be best practice; however, it works nonetheless.

```
useEffect(() => {
    const getNFTs = async() => {
        const nfts = await web3.alchemy.getNfts({
          owner: user.address,
          contractAddresses: [ contractAddress ],
          withMetadata: true
        })

        let modifiedData: any = []
        for (const nft of nfts.ownedNfts) {
          const tokenID = parseInt(nft.id.tokenId, 16)
          const image = `https://ipfs.io/ipfs/${nft.metadata?.image?.substring(5)}`

          modifiedData.push({
            contractAddress: contractAddress,
            tokenID: tokenID,
            name: nft.metadata?.name,
            image: image,
          })
        }

        setNFTs(modifiedData)
    }

    if (user.address) {
      getNFTs();
    }
}, [user])
```

Figure 4.19: getNFTs

A modal will appear for each NFT the user has if they click on the Verify NFT button, which has a text box for the secret code. The token Id is retrieved from the user clicking on a particular NFT, so they do not need to specify the token Id themselves. Once the secret code is entered, the user can click the validate button, which will call the proveOwnership method, which prompts a transaction for the user to confirm. The same code as the purchase NFT method is used as shown in Figure 4.18; however, the proveOwnership method is called instead of the safeMint method.

4.3.6 Viewing Event Logs

The View Event Logs button was created on the home page for viewing event logs. Clicking this page brings the user to the event logs page. The logs display in a neat table with all the headings and values clearly defined, allowing the bouncer to verify the details quickly. We now have the requirements of the dApp implemented successfully to abstract the functionality of the smart contract, making it easier for a user to utilise.

```
useEffect(() => {
    const getEventLogs = async() => {
      const provider = new ethers.providers.Web3Provider(window.ethereum);
      const signer = provider.getSigner();

      const contract = new ethers.Contract(
        user.address,
        abi,
        signer
      );

      contract.on("OwnershipApprovalRequest", (address, tokenId, secret) => {
        console.log(`Address: ${address}, tokenId: ${tokenId}, secret: ${secret}`);
      });

      let eventFilter = contract.filters.OwnershipApprovalRequest()
      let blockNumber = await provider.getBlockNumber()
      let events: any = await contract.queryFilter(
                        eventFilter, blockNumber-3000, blockNumber)

      setEvents(events)
    }

    if (user.address) {
      getEventLogs();
    }
  }, [])
```

Figure 4.20: Event Logs Code

4.3.7 Deploying the Dapp

To deploy the dApp Vercel was used. A Vercel account and linked to the GitHub account with the NFT dApp. Once it was linked, the GitHub repo for the dApp was selected. Vercel supports deploying Next.js apps. Once the build was completed, the dApp was deployed to https://nft-ticket-app.vercel.app/ for free with no setup required.

5

Walkthrough

The walkthrough chapter goes through the steps involved for all users using the system with respect to the requirements. The first section walks through the minimal solution using the blockchain explorer as a frontend. The second section will go through the walkthrough with the dApp solution to highlight and contrast the difference. The walkthrough helps in visualising the complete system, and also aids in the evaluation.

5.1 Minimal Solution

5.1.1 Purchasing a ticket

For a concert-goer to purchase a ticket for an event they wish to go to, they navigate to the smart contract address on the blockchain explorer, and click on the write contract tab (**?**). Here the concert-goer can connect their wallet by clicking Connect to Web3, and call the safeMint method with their address as the argument. This will initiate the transaction in their Metamask wallet, which they can confirm, and once the transaction is confirmed on the blockchain, they have successfully purchased the ticket, and they can view it on the blockchain explorer at their address.

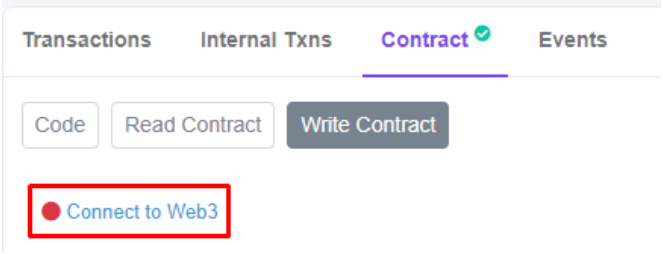

Figure 5.1: Connecting to PolygonScan

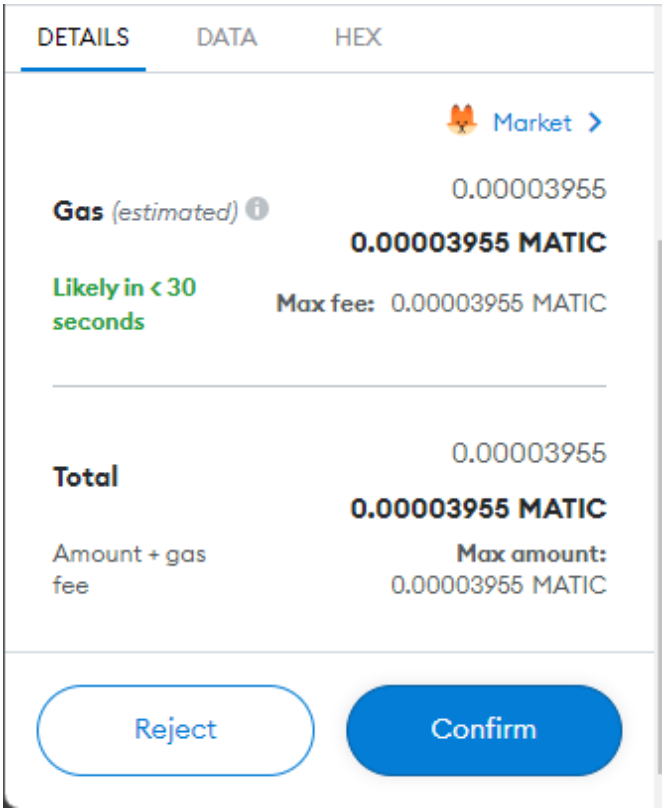

Figure 5.2: Confirming the Transaction

5.1.2 Validating the Ticket

Imagine the concert-goer has arrived at the venue, and they in-front of a bouncer who has requested they verify their ticket so they can let the concert goer into the venue. The bouncer communicates with the concert-goer a secret code as a form of two-factor authentication. Using the Metamask mobile app, the concert-goer

navigates to the smart contract address of the NFT Ticket (**?**). They then call the proveOwnership method with the tokenId of the NFT ticket they are trying to claim ownership of, and enter the secret code the bouncer has communicated to the concert-goer. If the transactions confirms, the OwnershipApprovalRequest event will be emitted onto the blockchain with the address, tokenId and secret code the concert-goer entered. If this matches what the bouncer communicated, the bouncer can view the event logs on the blockchain explorer and grant entry to the concert-goer into the venue.

Figure 5.3: proveOwnership Method on PolygonScan

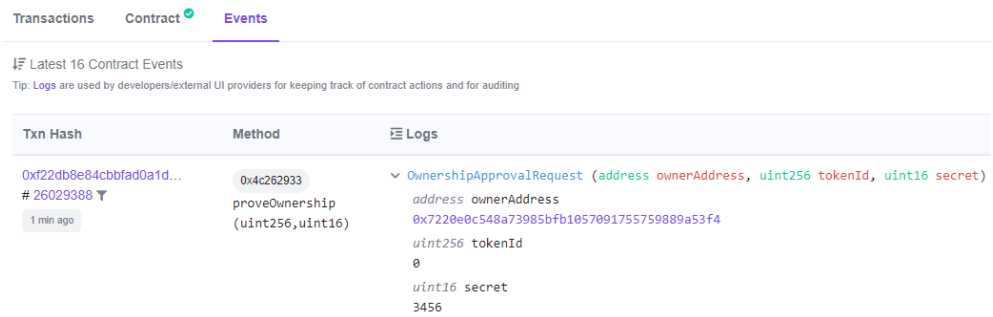

Figure 5.4: Smart Contract Event Logs

5.1.3 Trading the Ticket

The NFT can be traded by the concert-goer on any marketplace that supports Polygon NFTs like OpenSea as the ticket is an ERC-721 NFT. The concert-goer connects their wallet to OpenSea which will show the NFTs they own, and they can list their NFT ticket for sale, and anybody can purchase this NFT, and be able to verify the origin of the NFT, as it can be seen on the blockchain. The concert-goer's address and the NFT ticket collection can be viewed on OpenSea too through this link (OpenSea 2022)

5.2 Dapp Solution

To connect or sign in to the dapp, there is a sign in button located in the navbar for all pages of the dapp. The concert-goer clicks this sign in button on their mobile device, and will get a pop-up for Metamask asking to connect their wallet. Once connected, the concert-goer can use the dApp and is stayed logged in. If the concert-goer is connected to the wrong network, the dApp will display that they are connected to the wrong network, and ask them to click to switch network. This will trigger a Metamask pop up asking to switch networks.

5.2.1 Purchasing the ticket

To purchase the ticket using the dApp, the concert-goer can visit the purchase page of the dApp. They can click on the purchase ticket button which will call the safeMint method from the NFT ticket smart contract. This will trigger a pop up for the transaction, and the concert-goer can confirm it to buy the ticket. Once confirmed the ticket has been bought. The dApp also provides real time notifications to show the status of the transaction at the bottom right of the screen on desktop and at the bottom on mobile. When the purchase ticket button is clicked, the notification will display the the transaction has started and that the concert-goer must confirm the transaction. Once confirmed, a notification will state that the transaction has started and that it should confirm soon, and finally a notification displays that the transaction confirmed. Metamask provides notifications to show the status of transactions, however this system provides

more information on the status of the transaction, to help the user know exactly what is happening. Any errors in the transaction will also be caught and displayed by the notification system.

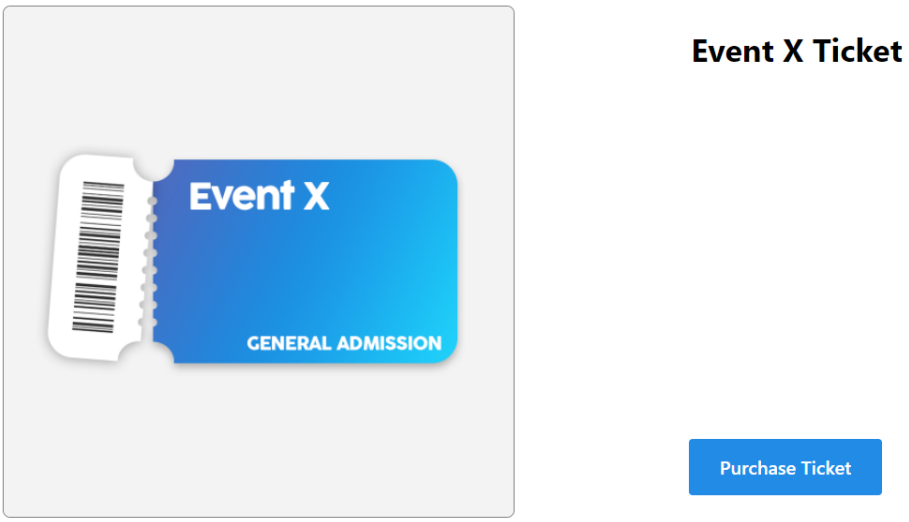

Figure 5.5: Purchase Ticket page

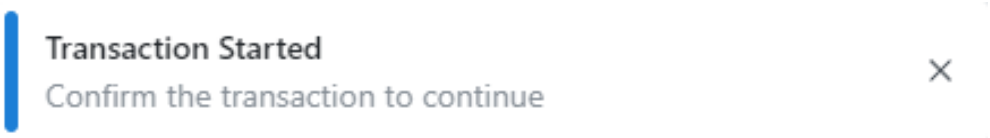

Figure 5.6: Transaction Started

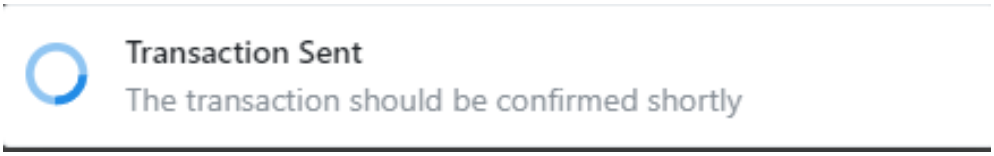

Figure 5.7: Transaction Sent

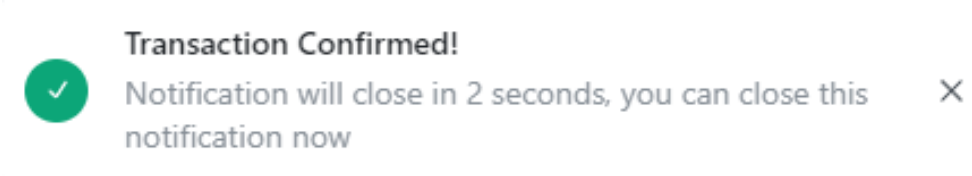

Figure 5.8: Transaction Confirmed

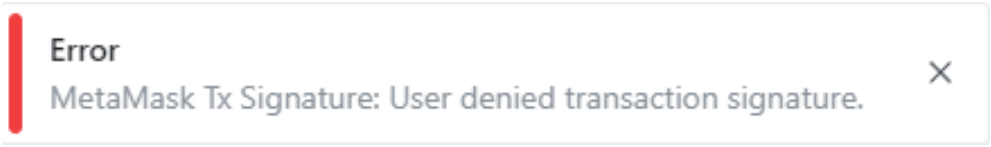

Figure 5.9: Transaction Error

5.2.2 Validating the Ticket

To validate the ticket, the concert-goer uses their mobile Metamask browser, and ensures they are connected to the dApp. Once connected they can visit the My Tickets tab which will display a list of the NFTs owned by the concert-goer. They can click the Verify Ticket button which opens a modal, requesting the secret code. They then enter the secret code and click the Verify button. This will call the proveOwnership method, except with the dApp, the concert-goer does not have to enter their address, or tokenId as the dApp already knows that information. Once the proveOwnership transaction confirms, the bouncer can view the Event Logs, to see if the ticket has been verified. Event Logs also has three icon buttons which represent the states verified, invalid and not checked. If the green check mark button is clicked, the table row will highlight green. The red examination mark will highlight the row red, and the last icon button will remove any row highlighting. This serves as a system to allow the bouncer to know if a ticket has been validated or not. The system can also check if previous tickets were marked as validated, by highlighting the row as yellow or another colour to indicate that the ticket has already been validated.

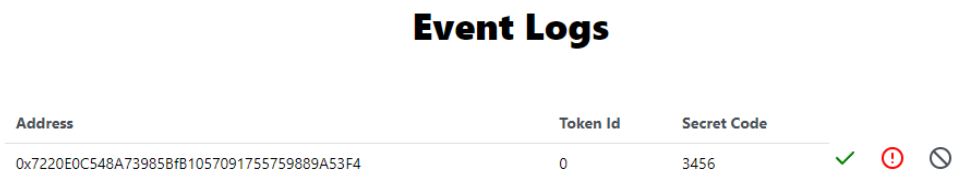

Figure 5.10: Event Logs

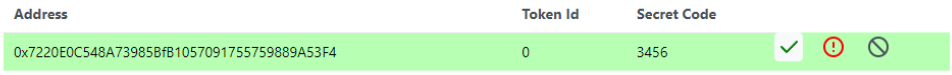

Figure 5.11: Event Logs Highlighted

5.2.3 Trading the Ticket

Trading the ticket is the same as the minimal solution.

6

Evaluation

The evaluation chapter aims to break down different characteristics that have to be considered when designing an event ticketing system and evaluate the event ticketing systems by comparing these characteristics to each other. The evaluation will compare the NFT ticket solution and current ticketing solutions. It is quite difficult to estimate the true development costs, however generalised observations can be made about these costs. The future work chapter will describe what can be done to improve the solution where it is more limited.

6.1 Ease of Use

To evaluate the ease of use of the solution, we have to evaluate the ways the user can use the system: purchasing the ticket, trading the ticket, verifying the ticket, and the pre-requisites the user needs to interact with the system.

6.1.1 Pre-requisites

For our solution, the pre-requisites are a mobile crypto wallet with sufficient cryptocurrency to buy the ticket. The NFT solution is unique, as the majority of people are not familiar witg the Web3 ecosystem. A first time user would have to download and install Metamask or their mobile device and or browser (a significant ramp up), whereas Web2 solutions would utilise an email for authentication.

Without email, another means of communication is needed with the customer. A notification or chat system would have to be implemented into the solution.

6.1.2 Purchasing the ticket

Purchasing MATIC to purchase the ticket is something that is not easy to do for first-time users as well. Most crypto exchanges require a KYC process to purchase crypto, which is a lengthy process of submitting an identification document like a passort along with a selfie to prove your idenity.

6.1.3 Trading the ticket

Users can trade their tickets on a marketplace like OpenSea. They allow users to list NFTs for sale for any price, and if it sells, the user receives the amount they listed it for minus fees. There is a secure, open marketplace for all tickets out of the box. Tickets can also be traded peer to peer using the smart contract transfer method. The drawbacks are that OpenSea can be tricky and confusing, especially with listing items and changing the price of items, gas fees, and signing transactions.

6.1.4 Verifying the Ticket

Current ticket solutions generally offer a QR or barcode ticket. A concert-goer arrives at the venue with a barcode for their ticket, either on paper or mobile, and the bouncer will scan the ticket to ensure it is valid and let the concert-goer into the venue. With the NFT solution, the bouncer tells the concert-goer a code, and that code is entered into the dapp; the user clicks the verify button, waits two seconds for the transaction to confirm, and then waits for the bouncer to see the event log and let concert-goer into the venue. The NFT solution is relatively slow compared to the current solution, and a user could enter the secret code wrong. The solution with the event logs is not scalable as a venue with multiple bouncers could have difficulty verifying tickets. As a minimum viable product, the solution works well but still has many improvements to be more usable without compromising security.

6.1.5 Speed

Polygon's average block processing time is 2.1 seconds. Transactions are bundled into blocks. So when a transaction is sent, like verifying a ticket or purchasing a ticket, it takes 2.1 seconds to confirm that transaction (Investopedia 2022). For purchasing tickets, 2.1 seconds is adequate as from personal experience, card payments generally take a few seconds to confirm. However for validating the ticket, this time could add up and slow down the queue system for entering venues.

6.2 Transaction Costs

We have to consider the development and server costs for the cost of ticketing systems. To develop an event ticketing system, we need a backend that handles the creation of tickets, stores user information in a database, accepts payments, verifies that tickets are valid, and handles authentication.

6.2.1 NFT Solution

The costs for the NFT solution would be servers for the dapp frontend and the development costs. The front end cost would be servers to send the web pages to the client. The development costs would include the one time cost of deploying the dapp and resources for developing the smart contract and dapp. The costs for the blockchain side of the dapp can be reviewed by using the hardhat-gas-reporter plugin. This plugin allows us to view the cost of deploying the smart contract, and the cost of the write methods like purchasing, transferring and proving ownership of the ticket. After running our tests, the plugin will display the cost of the tests in the terminal. Gas refers to the unit that measures the computational effort required to execute specific operations on the Ethereum network. Since each Ethereum transaction requires computational resources to execute, each transaction requires a fee. Gas refers to the fee required to conduct a transaction on Ethereum successfully (Ethereum 2022*f*). Essentially what this means is that different methods cost a different transaction fee. We can see the cost of executing our methods in Figure 6.1.

Solc version: 0.8.4	Optimizer enabled: false	Runs: 200	Block limit: 30000000 gas	
Methods				
Contract	**Method**	**Min**	**Max**	**Avg**	**# calls**	**eur (avg)**
Ticket	proveOwnership	-	-	26367	2	-
Ticket	safeMint	-	-	123679	3	-
Ticket	transferFrom	-	-	72078	1	-
Deployments				**% of limit**		
Ticket		-	-	3669715	12.2 %	-

Figure 6.1: hardhat-gas-reporter plugin

For Figure 6.1, the methods describe the methods that were executed. The 34 gwei/gas refers to the current gas prices. This number is how much base gas must be paid to get the transaction confirmed. This price fluctuates based on traffic on the blockchain. If many people are using the blockchain simultaneously, the gas price will be higher as the gas prices are paid to miners to confirm the transaction. The more gas paid, the faster the transaction will be confirmed. The formula for calculating the gas fees for a transaction is Gas units (limit) * Gas price per unit. Our gas units for safeMint is 123,679 and the gas price is 34 gwei. 123,679 * 34 is 4,205,086 gwei which translates to 0.0042 MATIC (Polygon's native crypto currency). The price of MATIC currently is 1.49 euro for 1 MATIC meaning that 0.0042 MATIC costs 0.006 euro to purchase a ticket. This price is the cost of the blockchain backend to purchase and store an NFT ticket on the blockchain forever. We then have a fully scalable backend and database deployed with no additional costs, and our NFT tickets are functional.

The dapp makes the system usable for the user, which is needed for real-world use. Alchemy provides nodes for us to communicate with the blockchain. Their costs are calculated in compute units. So we are charged for how much computing power we use rather than a single fixed cost for a server. The Alchemy documentation page (Alchemy 2022) shows the cost of different methods to the Ethereum blockchain. A common method that the dapp will use is eth_sendRawTransaction which is used for purchasing and verifying the ticket which costs 250 compute units. Other methods are used like eth_estimateGas that are used too for transactions, so let us say our cost per user is around 500 compute units in total. On

the free tier, we get 300 million compute units a month. We can call this method 600 thousand times, meaning 300 thousand tickets can be purchased and verified for free. To upgrade the usage, it costs 50$ a month for 400 million compute units, then it costs 1.2$ for every additional million compute units.

The costs of the backend are minuscule. The cost can be covered through a service fee or ticket profits. There is also a gas fee that the user has to pay for purchasing or verifying the ticket. The smart contract can cover this fee by implementing a gas station network, so the smart contract pays the cost of the gas fees rather than the user. This gas fee can also be taken from the ticket cost and based on the estimates above would cost about one cent in euro. Users have no card processing fees as payments are made in crypto.

6.2.2 Current Solution

It is quite difficult to determine the costs of the costs of operating a ticket system, as this information is encapsulated, we are only presented with the consumer cost. Eventbrite is a popular ticketing solution for event organisers to organise their own events. From their pricing website, and my location their pricing is €0.49 + 4% per paid ticket and Professional pricing is €0.69 + 5.5% per paid ticket (EventBrite 2022).

The development costs would include a hosted server and database. The server and database would require a scaling solution to handle large loads, and the database would need a backup solution in case data gets corrupted. This infrastructure would also have to be built by hiring a team of developers. Time can be saved by using external services like Firebase or MongoDB, which could further increase or decrease costs. Maintenance and testing of the backend also need to be considered. From this we can have a rough idea of the development costs involved.

6.3 Development Time

The development time for the NFT ticket solution is more weighted in terms of design than implementation. The dapp took the most significant amount of time

as it was slightly more challenging to connect Metamask and figure out the APIs. However, once that knowledge is known, the implementation is quite straight forward. The blockchain provides a backend, database and authentication system out of the box, with little coding and setup required for the implementation.

7

Conclusions and Future Directions

7.1 Conclusion

This project aimed to show that NFTs can be used for more than just pictures, and the underlying technology can be used to solve everyday problems and improve current technologies in different industries, in this case, the ticketing industry. The project research NFT ticketing solutions, designed and implemented an NFT ticketing solution while providing an in-depth inside into the process, and finally presented a walkthrough of the solution to help aid in the evaluation. There is currently a massive problem with verifying tickets and knowing if what a concert-goer is purchasing is an actual ticket. Millions of people have fallen victim to ticket scams and fraud, and the solution solved this issue by tracing the creation of the ticket to a smart contract on the blockchain. If the University of Limerick and other event organisers could implement NFT tickets as described, issues of ticket scammers would be solved. The details in the future work chapter would have to be implemented to make the solution usable in the real world. However, in terms of showing how NFT technology can solve these everyday problems, the project has done that.

7.2 Future Work

The future work chapter describes improvements that can be made based to improve the solution, based on the drawbacks uncovered in the evaluation chapter, many of which are to do with easier ramp-up and the user experience.

7.2.1 Use Email as a wallet

https://magic.link/ is a service that allows passwordless authentication. Users provide their email address and receive an email that they click on to log in. Magic is powered by crypto. Each account is tied to a wallet address on the Ethereum blockchain. Magic can be utilised just like Metamask, except onboarding is as easy as entering an email address and clicking a button. It provides an experience people are already familiar with and enriches the account as it is tied to an address. However, developers are locked into Magic's service, which may be a drawback. Pricing is 1,000 free active users and then 5 United States cents per user. An SDK is provided with docs on integrating Magic into a dapp easily. Wallets like Metamask can still be offered as a solution to log in. Magic has this feature coming soon also.

7.2.2 Ticket Purchasing

There are services like MoonPay which allow you to quickly buy crypto with a card instantly, without having to do a KYC process. MoonPay is also working on NFT integration, where users can purchase an NFT with a card. It would provide the same experience as current ticketing technologies that people are used to. Users would be allowed to pay with a card instead of crypto, making it easier for current internet users.

7.2.3 Verifying the Ticket

Verifying the ticket needs improvements in terms of ease of use. A QR code could be used, but this issue is being able to verify tickets while not compromising security.

7.2.4 Regulated Marketplace

A regulated marketplace for the tickets could be created and implemented into the smart contract and dapp to improve trading the ticket. A rough idea would be to allow the NFT only to be traded to the marketplace smart contract and disable trading of NFTs to other smart contracts, so peer to peer trading would still be permitted. The marketplace would be integrated into the dapp, and rules could be set for the marketplace like tickets are not allowed to be traded above sale price to stop ticket scalpers from buying and selling for a profit. After the event is over, the NFT could be allowed to trade on any marketplace then, by opening trade to all smart contracts. Ticketmaster offers a solution that allows users to sell their tickets on Ticketmaster, and people can buy them from there. However, this does not stop third-party marketplaces. This concept for a marketplace was implemented into the solution, however it was not planned as being part of the solution. The source code for the marketplace can be viewed at the public GitHub repos for the smart contract (**?**screpo), and dapp (**?**dapprepo). Every aspect of the concept described, was implemented.

References

abcoathup (2022), [online], available: https://forum.openzeppelin.com/t/how-to-verify-with-hardhat-or-truffle-a-smart-contract-using-openzeppelin-contracts/4119 [accessed: 12 April 2022] . 20

Actionfraud (2021), 'https://www.actionfraud.police.uk/news/beware-of-ticket-fraud-as-restrictionsease', available: https://www.actionfraud.police.uk/news/beware-of-ticket-fraud-as-restrictionsease [accessed: 2 January 2022] . 1

Alchemy (2022), [online], available: https://docs.alchemy.com/alchemy/documentation/compute-units [accessed: 15 April 2022] . 55

BAM (2022), [online], available: https://www.bam.fan/ [accessed: 3 January 2022] . 9

Blanco, J. (2022), [online], available: https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity [accessed: 12 April 2022] . 22

BlockchainHub (2022), [online], available: https://blockchainhub.net/blockchain-intro/ [accessed: 12 April 2022] . vi, 3, 4, 23

Blocknative (2022), [online], available: https://www.blocknative.com/blog/monitor-polygon-mempool#:~:text=On%20Polygon%2C%20the%20average%20block,transactions%20than%20the%20Ethereum%20network [accessed: 12 April 2022] . 14

Brechtpd (2022), [online], available: https://www.npmjs.com/package/base64-sol/v/1.0.1 [accessed: 15 April 2022] . 29

Coinbase (2022), [online], available: https://www.coinbase.com/learn/crypto-basics/what-is-polygon [accessed: 12 April 2022] . 14

REFERENCES

Dailyhodl (2018), 'Blockchain to Power Ticket Sales for Live Events During World Cup 2018', available: https://dailyhodl.com/2018/05/07/blockchain-topower-ticket-sales-for-live-events-during-world-cup-2018 [accessed: 3 January 2022] . 7, 8

Ethereum (2022*a*), [online], available: https://ethereum.org/en/developers/docs/web2-vs-web3/ [accessed: 19 February 2022] . 2

Ethereum (2022*b*), [online], available: https://ethdocs.org/en/latest/account-management.html?highlight=address#keyfiles [accessed: 12 April 2022] . 4

Ethereum (2022*c*), [online], available: https://eips.ethereum.org/EIPS/eip-875 [accessed: 3 January 2022] . 7

Ethereum (2022*d*), [online], available: https://ethereum.org/en/developers/docs/networks/ [accessed: 10 March 2022] . 23

Ethereum (2022*e*), [online], available: https://eips.ethereum.org/EIPS/eip-721 [accessed: 3 January 2022] . 29

Ethereum (2022*f*), [online], available: https://ethereum.org/en/developers/docs/gas/ [accessed: 10 March 2022] . 54

Etherscan (2022), [online], available: https://etherscan.io/ [accessed: 3 January 2022] . 19

EthersJS (2022), [online], available: https://docs.ethers.io/v5/ [accessed: 15 April 2022] . 35

EventBrite (2022), [online], available: https://www.eventbrite.ie/organizer/pricing/ [accessed: 25 April 2022] . 56

GET (2022), [online], available: https://www.get-protocol.io/ [accessed: 3 January 2022] . 8

GUTs (2022), [online], available: https://guts.tickets/ [accessed: 3 January 2022] . 8

Hardhat (2022*a*), [online], available: https://hardhat.org/ [accessed: 3 January 2022] . 20

Hardhat (2022*b*), [online], available: https://hardhat.org/tutorial/ [accessed: 3 January 2022] . 31, 32

Infura (2022), [online], available: https://infura.io/ [accessed: 15 April 2022] . 37

Investopedia (2022), [online], available: https://www.investopedia.com/polygon-matic-definition-5217569#citation-15 [accessed: 15 April 2022] . 54

IPFS (2022), [online], available: https://ipfs.io/ipfs/QmPtUx44pEg6KtorBrcjNxJYwx7S1nY7NPoCpmLUXxcBts [accessed: 15 April 2022] . 29

James T. Reese, J. and Thomas, D. (2013), Ticket Operations History and Background, *Ticket Operations and Sales Management in Sport*. 1

Komodo (2022), [online], available: https://komodoplatform.com/en/academy/bitcoin-wallet-address/#:~:text=The%20main%20difference%20is%20email,use%20the%20correct%20wallet%20address [accessed: 12 April 2022] . 4

Leonhardt, M. (2018), 'About 12 percent of people buying concert tickets get scammed', available: https://www.cnbc.com/2018/09/13/about-12-percent-of-people-buying-concerttickets get-scammed-.html [accessed: 2 January 2022] . 1

Metamask (2022*a*), [online], available: https://metamask.io/ [accessed: 3 January 2022] . 4, 18, 23

Metamask (2022*b*), [online], available: https://metamask.io/download/ [accessed: 12 April 2022] . 22

NextJS (2022), [online], available: https://nextjs.org/learn/basics/create-nextjs-app [accessed: 15 April 2022] . 35

Nodejs (2022), [online], available: https://nodejs.org/en/download/ [accessed: 12 April 2022] . 21

OpenSea (2022), [online], available: https://testnets.opensea.io/0x3e781ec0b18b75b5e3a8f23b79003bc7f1cf3954 [accessed: 15 April 2022] . 48

OpenZeppelin (2022), [online], available: https://docs.openzeppelin.com/contracts/4.x/wizard [accessed: 3 January 2022] . 26

REFERENCES

Oveit (2022), [online], available: https://oveit.com/nft-tickets/ [accessed: 3 January 2022] . 9

PolygonScan (2022), [online], available: https://mumbai.polygonscan.com/address/0xcaa7cfdd22c401db2addae7dc7a7cbd7fb84b260#code [accessed: 15 April 2022] . 34

Poylgon (2022), [online], available: https://docs.polygon.technology/docs/develop/network-details/network/ [accessed: 12 April 2022] . 24

PYMNTS (2019), 'How Ticketmaster Rewards Fans And Bars Fraudsters', available: https://www.pymnts.com/fraud-prevention/2019/how-ticketmaster-rewards-fans-bars-fraudsters-security/ [accessed: 2 January 2022] . 1

Secutix (2022), [online], available: https://www.secutix.com/customers/uefa [accessed: 3 January 2022] . 8

Solidity (2022), [online], available: https://docs.soliditylang.org/en/v0.8.13/abi-spec.html [accessed: 12 April 2022] . 39

TIXNGO (2022), [online], available: https://www.tixngo.io/ [accessed: 3 January 2022] . 9

Twitter (2022), [online], available: https://twitter.com/Victor928/status/1078456191276052481 [accessed: 3 January 2022] . 7

UEFA (2020), 'Blockchain to Power Ticket Sales for Live Events During World Cup 2018', available: https://www.uefa.com/insideuefa/mediaservices/news/025a-0f8e753e5b69-12cc5167a50a-1000--over-one-million-uefa-euro-2020-tickets-to-\be-distributedto-fa/ [accessed: 3 January 2022] . 8